

# Jegyzőkönyv

## Modern adatbázis rendszerek Msc

Féléves feladat

myBatis, mongoDB

**Készítette:** László Andrea

**Neptun kód:** DJ7PNE

**Dátum:** 2024. május 13.

# Tartalom

1.	ER modell és bemutatása.....	3
2.	MyBatis fejlesztőkörnyezet kialakítása .....	5
3.	Osztályok létrehozása az eclipse projektben .....	7
4.	Több-több kapcsolat megvalósítása myBatis-ben .....	8
5.	MongoDB környezet kialakítása .....	12
1.	Customer kollekció létrehozása, feltöltése egy, majd több adattal.....	12
2.	Factories kollekció létrehozása, feltöltése egy, majd több adattal .....	14
3.	Idegenkulcs mezők hozzáadása.....	14
4.	Idegenkulcs mezők kitöltése .....	15
5.	A gyárhoz egy új mező hozzáadása állapot néven .....	15
6.	A 2004 előtt alapított gyárak állapotának „túl régire” állítása .....	15
7.	„Túl régi” állapotú gyárak törlése .....	15
8.	Azon vevők kilistázása, melyeknek több, mint egy kapcsolata van gyárral .....	15
6.	Function-ek használata a mongoDB-ben.....	16
1.	Új function, megadott adatokkal való factory létrehozásához .....	16
2.	Új function, customer létrehozására, idegenkulcs megadásával is.....	16
3.	Tárolt függvény arra, hogy megadott origin alapján kikeresse az adott customereket .....	17
4.	Function, amely kiírja mely customerek tartoznak az egyes gyárhoz .....	18
5.	Minden factoryhoz adjunk hozzá egy mezőt "price" néven, ami a felbecsült értéket jelzi .....	18
6.	Function, amely a "túl régi" állapotú gyárak árát 25%-al csökkenti .....	18
7.	Számoljuk meg azon gyárak számát, mely ára kevesebb, mint 500000.....	19
8.	Írassuk ki place szerint az átlagárakat a factory collectionnál.....	19
9.	Hány factory rendelkezik helyenként "tul_regi" státusszal.....	19
10.	Scriptek betöltése .....	20
11.	Functionek tesztelése .....	20
7.	MongoDB használata Eclipsen belül.....	20
1.	Kliens csatlakozása a szerverhez.....	20
2.	Dokumentum elérése, adatbázis megnyitása.....	20
3.	Első elem kiválasztása és nevének kiírása a collection-ből.....	20
4.	Collection elemeinek kiírása .....	20
5.	Egy factory elem hozzáadása a collection-höz.....	21
6.	Több factory elem hozzáadása .....	21
7.	Azon elemek kiírása, amelyeknek felbecsült ára kevesebb, mint 500000.....	21
8.	A „túl régi” állapotú gyárak törlése.....	21
9.	A kliens lezárása.....	22

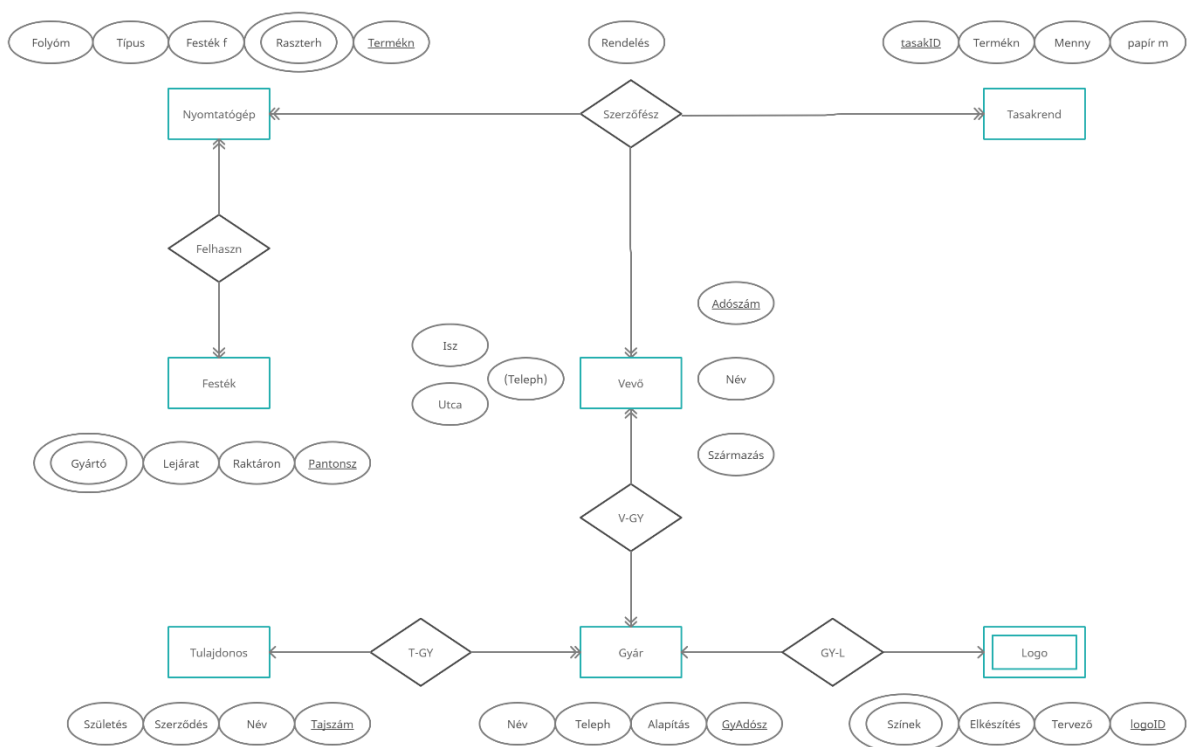
# 1. ER modell és bemutatása

Az ER modell egyes elemei:

- *Nyomtatógép egyed tulajdonságai*
  - Termék neve: A nyomtatógép egyedi kulcsa
  - Típus: A nyomtatógép típusa
  - Folyóméter: Az adott termékből nyomtatógép által nyomtatott mennyiség
  - Raszterhenger: Milyen fajta hengert használtak egyes színekhez
  - Festék felhasználása: Adott festék folyóméterenkénti felhasználása
- *Festék egyed tulajdonságai*
  - Pantonszám: A festék egyedi tulajdonsága
  - Gyártó: Adott festék gyártói
  - Raktáron lévő mennyiség: Éppen a gyárnál mekkora mennyiség van adott festékekből
  - Lejárat dátum: A festék lejáratainak dátuma
- *Tasakrendelés egyed tulajdonságai*
  - tasakID: A tasak egyedi kulcsa
  - Mennyiség: Mekkora mennyiséggel rendeltek (kg)
  - Termék neve: Adott termék neve, amelyet éppen rendelnek
  - Papír minősége: Milyen fajta papírból fogják gyártani az adott tasakot
- *Vevő egyed tulajdonságai*
  - Adószám: A vevő egyed kulcsa
  - Név: Vevő cég/személy neve
  - Telephely: Összetett tulajdonság, mely az irányítószámból és utcából áll
  - Származás: A vevő származási helye
- *Gyár egyed tulajdonságai*
  - Adószám: A gyár egyed kulcsa
  - Név: A gyár neve

- Telephely: A gyár telephelye, ahol elhelyezkedik
- Alapítási dátum: A gyár alapításának a dátuma
- *Logo egyed tulajdonságai*
  - Tervező: A logo tervezője
  - Színek: A logóban felhasznált színek
  - logoID: A verziószám, amely el lett fogadva az adott logóból
  - Elkészítés dátuma: A logo elkészítésének dátuma
- *Tulajdonos egyed tulajdonságai*
  - Tajsám: A tulajdonos egyedi kulcsa
  - Név: A tulajdonos neve
  - Születési dátum: Tulajdonos születési dátuma
  - Szerződés kezdete: Az a dátum, amikor tulajdonossá vált

Az ER modell:



## 2. MyBatis fejlesztőkörnyezet kialakítása

Alakítsuk ki a megfelelő fejlesztőkörnyezetet a myBatis használatához.

Először is töltsük le a myBatis jar-t és adjuk hozzá ezt az eclipse projektünkhöz.

[https://jar-download.com/?search\\_box=mybatis](https://jar-download.com/?search_box=mybatis)

(project properties -> java build path -> libraries -> add external jar)

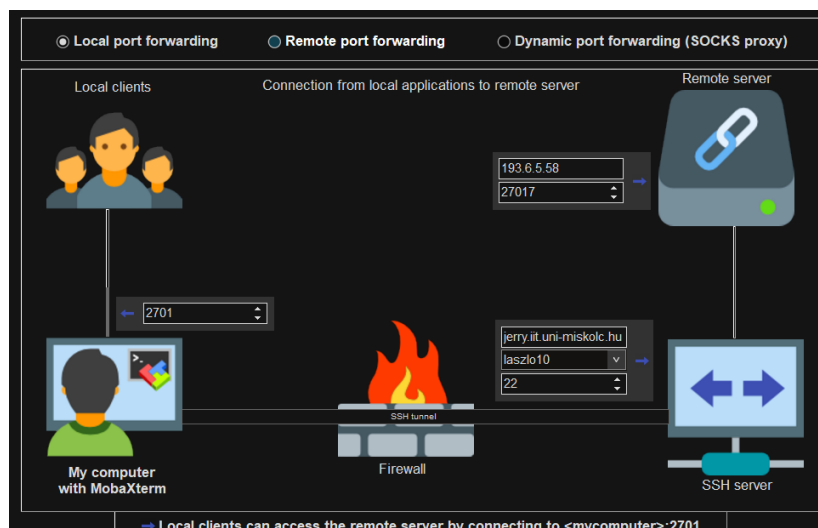
Ezek mellett szükséges még az SQL workbench letöltése is.

Érdeemes még az eclipse marketplace segítségével a „myBatis Generator”-t is installálni.

<https://dbschema.com/jdbc-driver/mysql.html>

A connectionok létrehozásához kell még a mysql jar is.

A mobaXTerm használatával alakítsunk ki egy tunneling kapcsolatot az alábbi értékekkel:



A jerry.iit cím alá a saját bejelentkezési felhasználónevünket kell megadni, majd elindítani ezt a kapcsolatot.

Mindezek után ha ezeket beállítottunk hozzunk létre egy eclipse projektet. Ezek után pedig a

file -> new -> other -nél írjuk be a myBatis generátort és generáltassunk vele egy

„mybatis\_config\_book.xml” fájlt, aminek így kell kinéznie az átalakítások után:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
  <typeAliases>
    <typeAlias type="com.concretepage.Book" alias="book"/>
  </typeAliases>
  <environments default="development">
    <environment id="development">
      <transactionManager type="JDBC"/>
      <dataSource type="POOLED">
        <property name="driver" value="com.mysql.cj.jdbc.Driver"/>
        <property name="url" value="jdbc:mysql://localhost:3306/book"/>
      </dataSource>
    </environment>
  </environments>
</configuration>
```

```

        <property name="username" value="root"/>
        <property name="password" value="*****"/>
    </dataSource>
</environment>
</environments>
<mappers>
    <mapper resource="com/concretepage/BookMapper.xml"/>
</mappers>
</configuration>

```

A username és password helyére a sajátunkat kell beírni, amely segítségével az sqlworkbenchben is létre tudunk hozni egy kapcsolatot.

Ha ez sikeres hozzunk létre az src mappában egy „com.concretepage” csomagot.

Ezalatt hozzunk létre egy BookMapper.xml fájlt, aminek a következőképpen kell kinéznie:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
    <mapper namespace="com.concretepage.BookMapper">
        <resultMap id="bookResult" type="book">
            <id property="isbn" column="isbn"/>
            <result property="title" column="title"/>
            <result property="price" column="price"/>
        </resultMap>

        <select id="findAllBooks" resultType="book" resultMap="bookResult">
            SELECT isbn, title, price FROM books
        </select>

        <insert id="insertBook" parameterType="book" keyProperty="isbn">
            INSERT into books(isbn, title, price)
VALUES(#{isbn},#{title},#{price})
        </insert>

    </mapper>

```

Létre kell még hoznunk egy táblát a könyveknek az sqlworkbench segítségével.

Hozzunk létre egy új connectiont miután telepítettük az sql adatbázist is, ezután az alábbi paranccsal hozzuk létre a táblát, majd töltjük fel néhány értékkel:

```
CREATE DATABASE book;
```

```
USE book;
```

```
CREATE TABLE books (isbn VARCHAR(14) PRIMARY KEY, title VARCHAR(50), price INT);
```

```
INSERT INTO books VALUES ("I444", "Első könyv", 5540);
```

```
INSERT INTO books VALUES ("I445", "Második könyv", 5740);
```

```
SELECT * FROM books;
```

### 3. Osztályok létrehozása az eclipse projektben

Hozzunk létre a „com.concretepage” mappa alatt egy Book osztályt, isbn, title és price változókkal.

Generáltassunk nekik getter, settereket.

Mindezek után hozzunk létre egy MyBatisUtil.java nevű osztályt, ami kezelni fogja a sessioneket.

```
public class MyBatisUtil {

    private static SqlSessionFactory sqlSessionFactory;
    static {
        String resource = "com/concretepage/mybatis_config_book.xml";
        InputStream inputStream;
        try {
            inputStream = Resources.getResourceAsStream(resource);
            sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static SqlSessionFactory getSqlSessionFactory() {
        return sqlSessionFactory;
    }

}
```

Hozzunk létre egy BookDao osztályt is, amely kezelni fogja a különböző műveleteket.

```
public class BookDao {

    public void save(Book book) {
        SqlSession session =
MyBatisUtil.getSqlSessionFactory().openSession();
        session.insert("com.concretepage.BookMapper.insertBook", book);
        session.commit();
        session.close();
    }

    public List<Book> getAllData(){
        SqlSession session =
MyBatisUtil.getSqlSessionFactory().openSession();
        List<Book> books =
session.selectList("com.concretepage.BookMapper.findAllBooks");
        session.close();
        return books;
    }

}
```

Végül hozzuk létre a futtatható osztályunkat is.

```
public class App {
```

```

public static void main(String[] args){

    BookDao bookDAO = new BookDao();

    //insert
    Book book1 = new Book();

    book1.setIsbn("I1");
    book1.setTitle("I1 könyv");
    book1.setPrice(4000);

    bookDAO.save(book1);

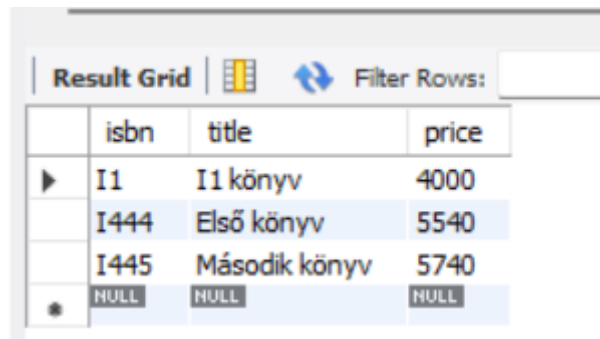
    //getAllData

    List<Book> books = bookDAO.getAllData();
    for(Book b : books) {
        System.out.println("isbn: " + b.getIsbn() + ", title: " +
b.getTitle() +
                                ", price: " + b.getPrice());
    }

}
}

```

Ezek után tudjuk ellenőrizni a mysql workbench segítségével, hogy az insert ott is sikeres volt.



	isbn	title	price
▶	I1	I1 könyv	4000
	I444	Első könyv	5540
	I445	Második könyv	5740
•	NULL	NULL	NULL

Mindenféle műveletet meg lehet valósítani, csak azt meg kell írni a mapper fájlba, illetve utána a DAO-ban is le kell kezelni.

## 4. Több-több kapcsolat megvalósítása myBatis-ben

Először is, ahhoz, hogy sikeresen működjön a program, mint az előző példában itt is létre kell hozni egy config.xml fájlt:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <typeAliases>
        <typeAlias type="com.database.Customer" alias="customer"/>
        <typeAlias type="com.database.Factory" alias="factory"/>
    
```



```

    <typeAlias type="com.database.CF" alias="cf"/>
</typeAliases>
<environments default="development">
    <environment id="development">
        <transactionManager type="JDBC"/>
        <dataSource type="POOLED">
            <property name="driver" value="com.mysql.cj.jdbc.Driver"/>
            <property name="url"
value="jdbc:mysql://localhost:3306/packet_factory"/>
            <property name="username" value="root"/>
            <property name="password" value="Andika2001"/>
        </dataSource>
    </environment>
</environments>
<mapper>
    <mapper resource="com/database/CustomerMapper.xml"/>
    <mapper resource="com/database/FactoryMapper.xml"/>
    <mapper resource="com/database/CFMapper.xml"/>
</mapper>
</configuration>

```

Ebben a config fájlban már megtalálható a később létrehozásra kerülő 3 mapper file elérése is.

Ezeket az előző példának megfelelően létre kell hozni, és meg kell benne adni a megfelelő lekérdezéseket.

A CFMapper file köti össze a Customer és Factory egyedet, annak segítségével lehet megoldani ezek összekapcsolását.

A CFMapper.xml fájl így néz ki:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
    <mapper namespace="com.database.CFMapper">
        <resultMap id="cfResult" type="cf">
            <result property="FTaxnumberCF" column="FTaxnumber"/>
            <result property="TaxnumberCF" column="Taxnumber"/>
        </resultMap>

        <resultMap id="factoryResult" type="factory">
            <id property="FTaxnumber" column="FTaxnumber"/>
            <result property="FoundationDate" column="FoundationDate"/>
            <result property="fName" column="fName"/>
            <result property="Premise" column="Premise"/>
        </resultMap>

        <select id="findAllFactoryCustomerRelationships" resultMap="cfResult">
            SELECT f.FTaxnumber, c.Taxnumber
            FROM Factory f
            INNER JOIN CF ON f.FTaxnumber = CF.FTaxnumberCF
            INNER JOIN Customer c ON CF.TaxnumberCF = c.Taxnumber;
        </select>

        <select id="findFactoriesWithCustomers" resultMap="factoryResult">
            SELECT DISTINCT f.*
            FROM Factory f
            INNER JOIN CF ON f.FTaxnumber = CF.FTaxnumberCF;
        </select>
    </mapper>

```

```

        <delete id="deleteFactoryRelationship" parameterType="long">
            DELETE FROM CF WHERE FTaxnumberCF = #{factoryId};
        </delete>

        <insert id="insertRelationship" parameterType="cf">
            INSERT into CF(FTaxnumberCF, TaxnumberCF)
            VALUES(#{FTaxnumberCF},#{TaxnumberCF})
        </insert>

    </mapper>

```

Mindezek után szükséges létrehozni az előzőekhez hasonlóan egy kezelő osztályt (MyBatisUtil.java), a megfelelő egyedeknek az osztályt, amelyek tartalmazzák a field-eket illetve azok getter, settereit, illetve ha szükséges konstruktorait (Customer.java, Favtory.java, CF.java). Szükséges egy külön osztályt még létrehozni, amely kezelni fogja az adatbázis kezeléséhez szükséges műveleteket, ez az AllDao.java, amely így néz ki:

```

public class AllDao {

    // customer

    public void saveCustomer(Customer customer) {
        SqlSession session =
MyBatisUtil.getSqlSessionFactory().openSession();
        session.insert("com.database.CustomerMapper.insertCustomer",
customer);
        session.commit();
        session.close();
    }

    public List<Customer> getAllCustomerData(){
        SqlSession session =
MyBatisUtil.getSqlSessionFactory().openSession();
        List<Customer> customers =
session.selectList("com.database.CustomerMapper.findAllCustomer");
        session.close();
        return customers;
    }

    public Customer getCustomerById(long customerId){
        SqlSession session =
MyBatisUtil.getSqlSessionFactory().openSession();
        Customer customer =
session.selectOne("com.database.CustomerMapper.findCustomerById", customerId);
        session.close();
        return customer;
    }

    // factory

    public void saveFactory(Factory factory) {
        SqlSession session =
MyBatisUtil.getSqlSessionFactory().openSession();
        session.insert("com.database.FactoryMapper.insertFactory", factory);
        session.commit();
        session.close();
    }
}

```

```

    public List<Factory> getAllFactoryData(){
        SqlSession session =
MyBatisUtil.getSqlSessionFactory().openSession();
        List<Factory> factories =
session.selectList("com.database.FactoryMapper.findAllFactories");
        session.close();
        return factories;
    }

    public Factory getFactoryById(long factoryId){
        SqlSession session =
MyBatisUtil.getSqlSessionFactory().openSession();
        Factory factory =
session.selectOne("com.database.FactoryMapper.findFactoryById", factoryId);
        session.close();
        return factory;
    }

    public void deleteFactoryById(long factoryId) {
        try (SqlSession session =
MyBatisUtil.getSqlSessionFactory().openSession()) {
            session.delete("com.database.FactoryMapper.deleteFactoryById",
factoryId);
            session.commit();
        }
    }

    // selections

    public List<CF> findAllFactoryCustomerRelationships() {
        SqlSession session =
MyBatisUtil.getSqlSessionFactory().openSession();
        List<CF> cf =
session.selectList("com.database.CFMapper.findAllFactoryCustomerRelationships");
        session.close();
        return cf;
    }

    public List<Factory> findFactoriesWithCustomers() {
        SqlSession session =
MyBatisUtil.getSqlSessionFactory().openSession();
        List<Factory> factory =
session.selectList("com.database.CFMapper.findFactoriesWithCustomers");
        session.close();
        return factory;
    }

    // relationship

    public void saveRelationship(CF cf) {
        SqlSession session =
MyBatisUtil.getSqlSessionFactory().openSession();
        session.insert("com.database.CFMapper.insertRelationship", cf);
        session.commit();
        session.close();
    }

    public void deleteFactoryRelationshipById(long factoryId) {

```

```

        try (SqlSession session =
MyBatisUtil.getSqlSessionFactory().openSession()) {
            session.delete("com.database.CFMapper.deleteFactoryRelationship",
factoryId);
            session.commit();
        }
    }
}

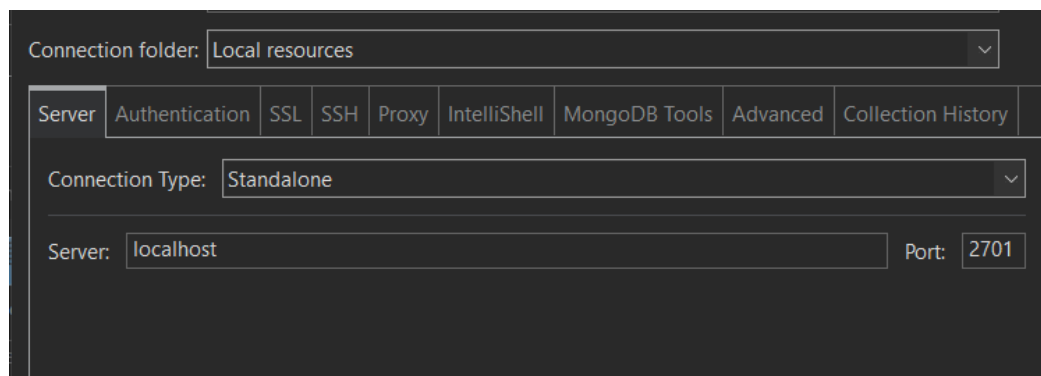
```

Mindezek után már csak egy futtatható osztályra lesz szükségünk, amelyben megnézhetjük a metódusok működését, illetve ezt az sqlworkbench-ben is tudjuk ellenőrizni.

## 5. MongoDB környezet kialakítása

A mobaXTerm kapcsolat elindítása most is szükséges, illetve a studio3T környezet telepítése is.

Miután elindítottuk a programot, egy új connectiont hozzá kell adnunk, ehhez a következő adatokat:



Ha sikerült csatlakoznunk a szerverhez adjunk hozzá egy új adatbázist a neptunkódunkkal.

Miután ezt megcsináltuk, nyissuk meg ezt az IntelliShellben, ahol dolgozhatunk tovább.

Hozzunk létre collection-öket, majd ezekhez megfelelő metódusokat.

```
//adatbázis kiválasztása
```

```
//USE "DJ7PNE";
```

### 1. Customer kollekció létrehozása, feltöltése egy, majd több adattal

```
db.createCollection("customers");
```

```
db.customers.insertOne({
```

```
    _id: "444",
```

```
    name: "customer",
```

```
    origin: "england",
    place: { postcode: "37624", street: "Buckingham" },
});
db.customers.insertMany([
  {
    _id : "000",
    name : "first customer",
    origin : "hungary",
    place : {
      postcode : "4090",
      street : "Abbey"
    }
  },
  {
    _id : "111",
    name : "second customer",
    origin : "hungary",
    place : {
      postcode : "4200",
      street : "London"
    }
  },
  {
    _id : "222",
    name : "third customer",
    origin : "germany",
    place : {
      postcode : "24124",
      street : "Frankfurt"
    }
  }
]);
```

## 2. Factories kollekció létrehozása, feltöltése egy, majd több adattal

```
db.createCollection("factory");
db.factory.insertOne({
  _id: "004",
  name: "f factory",
  place: "switzerland",
  foundation_date: "2010. 04. 23",
});
db.factory.insertMany([
  {
    _id : "001",
    name : "f factory",
    place : "hungary",
    foundation_date : "2020. 07. 03"
  },
  {
    _id : "002",
    name : "s factory",
    place : "england",
    foundation_date : "2000. 02. 09"
  },
  {
    _id : "003",
    name : "t factory",
    place : "england",
    foundation_date : "2004. 11. 14"
  }
]);
```

## 3. Idegenkulcs mezők hozzáadása

```
db.customers.updateMany({}, { $set: { factory_id: [] } });
db.factory.updateMany({}, { $set: { customer_id: [] } });
```

#### 4. Idegenkulcs mezők kitöltése

```
db.customers.updateOne({_id: "444"}, {$set: {factory_id: [004]}});  
db.customers.updateOne({_id: "000"}, {$set: {factory_id: [001, 002]}});  
db.customers.updateOne({_id: "111"}, {$set: {factory_id: [002]}});  
  
db.factory.updateOne({_id: "004"}, {$set: {customer_id: [444]}});  
db.factory.updateOne({_id: "001"}, {$set: {customer_id: [000]}});  
db.factory.updateOne({_id: "002"}, {$set: {customer_id: [111, 000]}});
```

#### 5. A gyárakhoz egy új mező hozzáadása állapot néven

```
db.factory.updateMany({}, { $set: { állapot: "uj" } });
```

#### 6. A 2004 előtt alapított gyárak állapotának „túl régire” állítása

```
db.factory.updateMany({ foundation_date: { $lt: "2004" } }, { $set: { állapot:  
"tul_regi" } });
```

#### 7. „Túl régi” állapotú gyárak törlése

```
db.factory.deleteMany({ állapot: "tul_regi" });
```

#### 8. Azon vevők kilistázása, melyeknek több, mint egy kapcsolata van gyárral

```
db.customers.aggregate([  
  {  
    $lookup: {  
      from: "factory",  
      localField: "_id",  
      foreignField: "customer_id",  
      as: "factoryList"  
    }  
  },  
  {  
    $project: {  
      _id: 1,
```

```

        name: 1,
        numberOfFactories: { $size: "$factoryList" }
    }
},
{
    $match: {
        numberOfFactories: { $gt: 1 }
    }
}
]);

```

## 6. Function-ek használata a mongoDB-ben

### 1. Új function, megadott adatokkal való factory létrehozásához

```

db.system.js.save(
    {
        _id: "add_factory",
        value: function (id, name, place, foundation_date, allapot){
            db.factory.insertOne({
                _id: id,
                name: name,
                place: place,
                foundation_date: foundation_date,
                allapot: allapot
            })
        }
    }
);

```

### 2. Új function, customer létrehozására, idegenkulcs megadásával is

```

db.system.js.save(
    {

```



```

    _id: "add_customer",
    value: function (id, name, origin, postalCode, street, factory_id){
        db.customers.insertOne({
            _id: id,
            name: name,
            origin: origin,
            place: {
                postalcode: postalCode,
                street: street
            },
            factory_id: factory_id
        })
    }
}
)

```

3. Tárolt függvény arra, hogy megadott origin alapján kikeresse az adott customereket

```

db.system.js.save(
{
    _id: "getCustomersByOrigin",
    value: function (origin){
        var q = db.customers.find({
            origin: origin
        });
        while(q.hasNext()){
            print(tojson(q.next()));
        }
    }
}
)

```

4. Function, amely kiírja mely customerek tartoznak az egyes gyárhoz

```
function printCustomersForFactory(factoryId) {
    db.factory.aggregate([
        { $match: { "_id": factoryId } },
        { $lookup: {
            from: "customers",
            localField: "customer_id",
            foreignField: "_id",
            as: "customers"
        }},
        { $unwind: "$customers" },
        { $project: { "factory_name": "$name", "customer_names":
"$customers.name" }}
    ]).forEach(function(doc) {
        print("A(z) " + doc.factory_name + " gyárhoz tartozó ügyfelek: " +
doc.customer_names);
    });
}
```

5. Minden factoryhez adjunk hozzá egy mezőt "price" néven, ami a felbecsült értéket jelzi

```
db.factory.updateMany({}, { $set: { price: 500000 } });
```

6. Function, amely a "túl\_régi" állapotú gyárak árát 25%-al csökkenti

```
db.factory.find().forEach(
    function(obj){
        if(obj.allapot == "tul_regi"){
            db.factory.update({
                _id: obj._id,
            }, {
                $mul : {price : 0.75}
            });
        }
    })
```

7. Számoljuk meg azon gyárak számát, mely ára kevesebb, mint 500000

```
db.factory.find({"$where": function(){
    if(this.price < 500000)
        return true;
    else
        return false;
}})
```

8. Írassuk ki place szerint az átlagárakat a factory collectionnél

```
db.factory.aggregate(
    {
        $group : {"_id": "$place", "avgprice" : {"$avg": "$price"}}
    }
)
```

9. Hány factory rendelkezik helyenként "tul\_regi" státusszal

```
db.factory.aggregate([
    { $match: { status: "tul_regi" } },
    { $group: { _id: "$type", count: { $sum: 1 } } },
    { $sort: { count: -1 } }
])

db.factory.aggregate([
    {
        $match: { "allapot": "tul_regi" }
    },
    {
        $group: {"_id": "$place", "db": { $sum: 1 }}
    },
    {
        $sort: {"db": -1}
    }
])
```

## 10. Scriptek betöltése

```
db.loadServerScripts();
```

## 11. Functionek tesztelése

```
add_factory("555", "inserted_with_function", "hungary", "2022.02.02",  
"új");  
  
add_customer("c111", "customer_name", "customer_origin", "12345", "Example  
Street", ["001", "002"]);  
  
getCustomersByOrigin("hungary");  
  
printCustomersForFactory("004");
```

# 7. MongoDB használata Eclipsen belül

Ahhoz, hogy a mongoDB-t tudjuk használni Java projektben le kell tölteni a mongo-java-driver jar file-t és hozzá kell adni a projekt build path-hoz. Miután ez megtörtént létrehozhatjuk a futtatható osztályunkat.

## 1. Kliens csatlakozása a szerverhez

Itt is szükséges lesz a mobeXTerm csatlakozáshoz, ahhoz, hogy sikeresen le tudjanak futni a kéréseink. Mindezek után szükséges egy MongoClient létrehozása, melynek segítségével a kapcsolatot meg tudjuk valósítani.

```
MongoClient mongoClient = MongoClient.create("mongodb://localhost:2701");  
MongoDatabase db = mongoClient.getDatabase("DJ7PNE");
```

## 2. Dokumentum elérése, adatbázis megnyitása

```
com.mongodb.client.MongoCollection<Document> table = db.getCollection("factory");  
Document doc = new Document();  
System.out.println("Adatbázis megnyitása és collection megnyitása  
kész");
```

## 3. Első elem kiválasztása és nevének kiírása a collection-ből

```
System.out.println("Első elem neve a collectionben");  
doc = table.find().first();  
String res = doc.getString("name");  
System.out.println(res);
```

## 4. Collection elemeinek kiírása

```
System.out.println("A collectionben lévő elemek");  
for (Document doc2 : table.find()) {  
    System.out.println(doc2.toJson());  
}
```

```
}
```

## 5. Egy factory elem hozzáadása a collection-höz

```
System.out.println("Egy factory elem hozzáadása");
Document newFactory = new Document("_id", 8)
    .append("name", "java inserted")
    .append("place", "germany")
    .append("foundation_date", "2022.06.30")
    .append("customer_id", 222)
    .append("allapot", "uj")
    .append("price", 400000);

table.insertOne(newFactory);
```

## 6. Több factory elem hozzáadása

```
System.out.println("Több factory elem hozzáadása");
Document newFactory2 = new Document("_id", 9)
    .append("name", "java inserted2")
    .append("place", "germany")
    .append("foundation_date", "1999.12.10")
    .append("customer_id", 444)
    .append("allapot", "tul_regi")
    .append("price", 232000);

Document newFactory3 = new Document("_id", 10)
    .append("name", "java inserted3")
    .append("place", "germany")
    .append("foundation_date", "2003.04.23")
    .append("customer_id", 111)
    .append("allapot", "tul_regi")
    .append("price", 600000);

ArrayList<Document> factories = new ArrayList<>();
factories.add(newFactory2);
factories.add(newFactory3);

table.insertMany(factories);
```

## 7. Azon elemek kiírása, amelyeknek felbecsült ára kevesebb, mint 500000

```
System.out.println("Azok az elemek, amelyeknek a felbecsült ára kevesebb, mint 500000");
MongoCursor<Document> cursor = (MongoCursor<Document>)
table.find(Lt("price", 500000)).iterator();
try {
    while(cursor.hasNext()) {
        System.out.println(cursor.next().toJson());
    }
} finally {
    cursor.close();
}
```

## 8. A „tul\_regi” állapotú gyárak törlése

```
System.out.println("A 'tul_regi' állapotú gyárak törlése");
```

```
        MongoClient<Document> cursor2 = table.find(eq("allapot",
"tul_regi")).iterator();
        try {
            while(cursor2.hasNext()) {
                Document docDel = cursor2.next();
                table.deleteOne(docDel);
            }
        } finally {
            cursor2.close();
        }
    }
```

## 9. A kliens lezárása

```
mongoClient.close();
```